DIAMANLAB – AN INTERACTIVE TAYLOR-BASED CONTINUATION TOOL IN MATLAB

Isabelle Charpentier[1,2], Bruno Cochelin[3] and Komlanvi Lampoh[3]

[1]Laboratoire d'Etude des Microstructures et de Mécanique des Matériaux, UMR CNRS 7239, Ile du Saulcy, 57045 Metz Cedex 1, France
[2]Laboratoire Icube, UMR 7357, 2 rue Boussingault, 67000 Strasbourg, France, isabellecharpentier@hotmail.com
[3]Laboratoire de Mécanique et d'Acoustique, UPR CNRS 7051, 31, Chemin Joseph-Aiguier, F-13402 Marseille cedex 20, France

September 20, 2013

Abstract. With the interactive continuation tool Diamanlab, solution branches of a parametric nonlinear problem are computed as sets of Taylor-based solutions stored in checkpoints. Theoretical aspects and implementation are generic, taking advantage of the efficient higher-order asymptotic numerical method in its Diamant form that interprets the generic nonlinear problem as a sequence of linear ones, of Automatic Differentiation (AD) for Taylor coefficient computations, of object-oriented programming and graphical user interface capabilities of MATLAB. The implementation involves four classes devoted to the interactive management of the continuation, to the manipulation of a generic system of nonlinear equations, to the checkpoint management and to higher-order AD, respectively.

In practice, any analytical nonlinear system of equation may be implemented in a natural way as a subclass of the generic system class, then solved in an easy manner using the graphical user interface. A benchmark of classical nonlinear problems is provided to serve as a basis for the implementation user-defined problems. Diamanlab usage and bifurcation detection are discussed on the Brusselator problem whose solution involves three interconnected loops. Additional user-defined graphics are presented for the Bratu problem.

Asymptotic numerical method, automatic differentiation, MANLAB, Diamant, graphical user interface

1. Introduction

Numerical continuation and bifurcation analysis of nonlinear equation solutions are classical numerical tools in many scientific areas. During the last thirties, general-purpose software, free and commercial ones, have been proposed to engineers and scientists to draw bifurcation diagrams without embarking into the risky and heavy task of programming their own continuation algorithm. Most of them rely on the first order predictor-corrector principles described in [1, 14, 21]. Software are nowadays provided with a Graphical User Interface (GUI) allowing interactions

Corresponding author: isabellecharpentier@hotmail.com

with the user. We refer to [15] for an overview of existing packages for continuation and bifurcation analysis. In any case, a good knowledge of the underlying continuation strategy is generally needed for a good tuning of the methodological parameters of such predictor-corrector algorithms.

The Asymptotic Numerical Method (ANM) is an alternative to first order predictor-corrector methods. Solution branches are approximated as higher-order truncated Taylor expansions for which the range of validity is estimated *a posteriori* from the remainder of the series. Under analyticity assumptions, this allows for an automatic and adaptive computation of the continuation step size that ensures the robustness of the method. Hence, no methodological parameter need to be tuned. The Matlab package MANLAB [2] provides an object-oriented implementation of the ANM as well as a GUI. It manages "automatic" series calculations for user-defined problems written in a quadratic formalism, what has constituted the main limitation to the dissemination of MANLAB as a general purpose continuation tool.

Automatic Differentiation (AD) [17] is the more practicable approach to higher-order differentiation, providing generality, efficiency and ease of use. Diamant, the AD version of the ANM [8, 6], computes series from the user's equation in a straightforward manner. The Diamanlab tool described in the paper combines the robustness, object-oriented programming and interactivity of the MANLAB tool with the AD abilities of Diamant. The implementation involves four classes devoted to the interactive management of the continuation, to the manipulation of a generic system of nonlinear equations, to the checkpoint management and to higher-order AD, respectively. Diamanlab relies on operator overloading as the vehicle of attaching higher-order derivative computations to the arithmetic operators and intrinsic functions provided by the programming language [3, 16, 20, 9]. A Diamanlab user implements his equation system as a classical Matlab function, sets the initial guess, and uses the GUI to plot his bifurcation diagram. Moreover, object-oriented programming allows for the implementation of user-defined graphics for particular analysis of the computed solution branches.

The layout of the paper is as follows. Fundamentals of ANM-based continuation tools, including Diamant, are briefly presented in Section 2. Section 3 discusses with detail the object-oriented implementation of the Diamanlab package. A few examples of interactive continuation usage are provided in Section 4. Finally, Section 5 provides a summary and an outlook.

2. ANM-based continuation tools

Let $R(U) = R(u, \lambda)$ define a nonlinear algebraic system comprising n equations, where u is a state vector of dimension n, λ is a scalar control parameter, and $U = (u, \lambda)$ for the sake of concision. Solutions of (1),

$$Find\ U = (u, \lambda) \in \mathbb{C}^n \times \mathbb{R}\ such\ that\ R(U) = 0, \tag{1}$$

are one-dimensional continua of solution points, called solution branches, which may intersect at bifurcation points.

Continuation algorithms constitute a classical answer to solution branch computations. Under differentiability assumption, the problem (1) may be solved through a first order predictor-corrector method that computes the solution branch as a collection of converged solution points [1, 14, 21]. To be effective, an elaborate strategy

is mandatory for a control of the step-size that guarantees a good compromise between the number of corrections per step and the size of each step.

Under analyticity assumption, higher-order order methods [22] such as the ANM [12, 13] provide solution branches as a collection of continuous parametric representations written as truncated Taylor expansions,

$$U(a) = \sum_{p=0}^{P} a^p \frac{1}{p!} \frac{\partial^p U}{\partial a^p}(0) = \sum_{p=0}^{P} a^p U_p, \quad \forall a \in (0, a_m), \tag{2}$$

where a is a path parameter, generally the pseudo-arc length parameter, a_m is the range of validity of the series in which $U(a)$ satisfies the accuracy required by the user, U_p is the unknown Taylor coefficient of U at order p, and P the truncation order. In the ANM, series (2) are introduced in the actual user-defined equations. This yields a sequence of linear problems sharing the same Jacobian matrix but different right-hand-side terms, which solution enables the computation of the sequence of $\{U_p\}_{p=1,..,P}$ in an iterative manner. The step size control strategy relies on an automated calculation of the range of validity a_m [10] from the remainder of the current series (2). The solution point $U(a_m)$ is used as initial guess to compute the next parametric representation (2). By construction, the ANM is a robust and efficient continuation process. A Newton-Raphson (NR) correction may be performed when the accuracy of the initial guess U_0 exceeds a user-prescribed value.

Great improvement in terms of generality is achieved through Diamant [6]. Series (2) are introduced in the generic problem (1). Following Faá di Bruno's generalization of the chain rule to higher-order derivatives [7, 19], this yields a sequence of P linear systems,

$$R_p = R_1 U_p + \{R_p | U_p = 0\} U_1 = 0, \quad p = 1, .., P, \tag{3}$$

where the Jacobian R_1 is the same over the orders. The higher-order derivative $\{R_p|U_p = 0\}$ is the Taylor coefficient R_p evaluated with the value of the unknown U_p set to 0. Following (3), the linear system at order $p \geq 1$ is

$$\begin{pmatrix} R_1 \\ U_1' A \end{pmatrix} U_p = \begin{pmatrix} -\{R_p|U_p = 0\} \\ \delta_{1p} \end{pmatrix}, \tag{4}$$

where δ_{1p} is the Kronecker's delta, U_1' the transpose of vector U_1, and A a matrix implementing some path equation [13].

From a computer point of view, the Jacobian and the higher-order terms are calculated applying AD on the residual function R provided by the user.

3. Implementation

Diamanlab is implemented for MATLAB version 7.0 or higher. No additional toolbox is necessary. Key aspects of the Diamanlab implementation are generality, efficiency and interactivity. Generality and efficiency are present at different stages taking advantage of Diamant, AD and object-oriented programming. First, any set of actual analytic nonlinear equations (1) is represented as a generic function named `R` in both the Diamant framework and its Matlab implementation. Second, the user's equations are coded deriving the class `UserSyst` from the class `Syst` of Diamanlab, see the class diagram Fig. 1. Third, the AD operator overloading library coded in the `Taylor` class allows for the automation of the Jacobian and

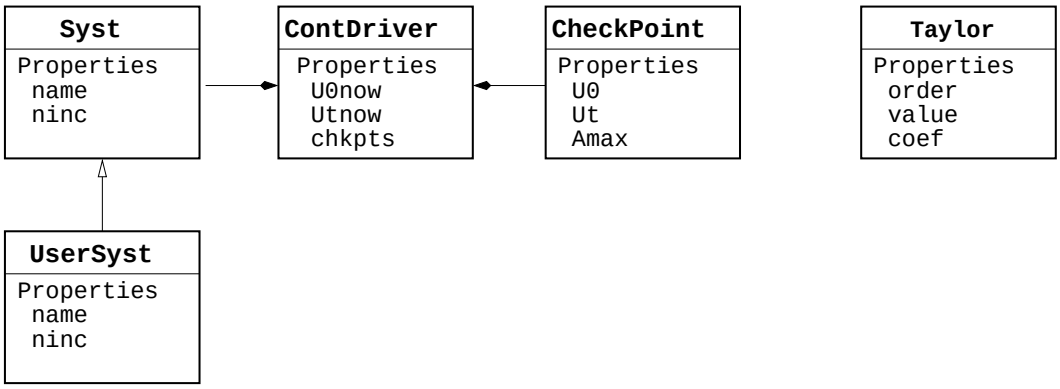

FIGURE 1. Class diagram and main properties

Taylor coefficient computations appearing in (4). The continuation is implemented in the `ContDriver` class meanwhile information about computed series are stored in `CheckPoint` objects. A particular attention is paid to class, method and property names which are indicated using typewriter characters. Interactivity is provided by a Graphical User Interface (GUI) and real time visualization tools for bifurcation analysis. In this section, classes are presented from the lower level ones, `Taylor` and `CheckPoint` for series representation, to the higher level ones, `Syst` and `ContDriver` dealing with the generic nonlinear system of equation.

3.1. **The `Taylor` class.** This class implements an operator overloading library devoted to higher-order AD. A `Taylor` object represents a truncated Taylor expansion by means of three properties: the truncation `order` P, a 2D `value` array and a 3D `coef` array containing U_0 and U_p for $p = 1$ to P, respectively. For the sake of generality, scalar variables are coded as 1×1 arrays. The Taylor constructor is designed to be called with a variable number of input arguments. The order is mandatory. Default array values are set to 0. Methods define how built-in operators and functions work on `Taylor` series objects. The current AD implementation is based on re-computations up to the current order p even operation count of Taylor-based series may be of $\mathcal{O}(P^2)$ when storing intermediate Taylor coefficients [4]. Diamanlab 1.0 does not include such opportunities for now. Meaningful operations and functions on Taylor series are overloaded. This includes math functions such as the exponential, the logarithm, trigonometric functions and vectorial functions (`norm`, `dot`, `sum`, `cat`), as well as array and matrix multiplications for which eventual dimension disagreements are raised. Particular `subsasgn` and `subsref` methods allow for the interpretation of indexed assignment statements involving `Taylor` objects. Some converters from `Taylor` to `double`, and vice versa, are implemented through the `get` and `set` methods.

3.2. **The `CheckPoint` class.** The ANM continuation generates a succession of branch sections, using the end point of each section as the new U_0 point for the next section. Here, this basic principle of the ANM is implemented using checkpoints [18, 5] designed to carry series information computed for particular solutions U_0 of (1), allowing for a restart of the computations or a stability analysis.

In Diamanlab 1.0, a `CheckPoint` object is the aggregation of a `Taylor` object `U0` containing the Taylor series computed at point U_0, a tangent vector `Ut` indicating a traveling direction, a range of validity `Amax` for the series `U0`, and some representations of the solution branch for plotting purposes. The `CheckPoint` class contains two methods only. The `eval` method provides a discrete "point-by-point representation" of the branch. The `endpoint` method sets the solution point U_{end} to $U(a_{max})$, the value of series evaluated at the limit of validity of `U0`, and the tangent direction U_{tend} pointing outward to $\frac{dU}{da}(a_{max})$.

3.3. **The `Syst` superclass and user-defined subclasses.** The `Syst` superclass provides methods related to the generic system (1) of nonlinear equations. A `Syst` object has two "problem-dependent" properties that should be provided at runtime: a `name` to be used in the plots, for instance, and the number of equations of the system, namely `ninc`. Other ones are methodological parameters, set to default values. The `Syst` methods are `Jacobian`, `tangentvector`, `NRcorrections` and `ANMseries`. The `Jacobian` method builds the Jacobian R_1 at point $U(0)$ using the canonical basis. This technical procedure is convenient for small problems only, sparse Jacobian evaluation will be implemented soon. The `tangentvector` method computes an oriented unitary tangent vector to the branch using this Jacobian. The `NRcorrections` method implements Newton-Raphson corrections to improve the accuracy of a solution if desired. Devoted to series computation, `ANMseries` is the key method of Diamanlab. Inputs are the `Syst` object related to the system under study and defined through a `Syst` subclass hereafter denoted by `UserSyst`, the Taylor variable `U0` which `value` property is known, and the oriented tangent vector `Ut`. Outputs are the updated series `U0` and the estimated range of validity `Amax`. `ANMseries` first calls the `Jacobian` and the `tangentvector` methods. Then, iterations are performed from order $p = 1$ to P to compute the Taylor coefficients R_p of R. Each of the iterations consists in the evaluation of the right-hand-side term $\{R_p|U_p = 0\}$ and the solution of system (4). The range of validity `Amax` is deduced from the computed series [13].

The system of nonlinear equations under study is implemented as a `UserSyst` class. This `UserSyst` class inherits properties and methods from the `Syst` class, and contains methods that are specific to the user's system. The `R` method coded in `UserSyst` defines the actual system of equation to be solved: the input is a Taylor objet U, the output is the AD computed Taylor object R to be used in the `Syst` methods such as `ANMseries`, `Jacobian`, and so forth. For the sake of generality, the `R` method result is a double vector when the input is a double vector. The `UserSyst` class may also contain some user displays to plot and to interpret results in a specific manner, detail is reported in paragraph 4.3. A benchmark of user's classes is included in Diamanlab.

3.4. **The `ContDriver` class and the GUI.** The `ContDriver` class is targeted to the implementation of the Diamant version of the ANM continuation process. Properties are the "current point" of continuation figured by the two variables `U0` and `Ut`, and a list of `CheckPoint` objects. Methods are mainly those indicated as pushbuttons on the GUI, Fig. 2. At execution time, Diamanlab launches the GUI and a Matlab figure to plot the projected bifurcation diagram.

The projected bifurcation diagram is a 2D plot of one or several curves, each of them showing the evolution of one component of U versus another. On each of the

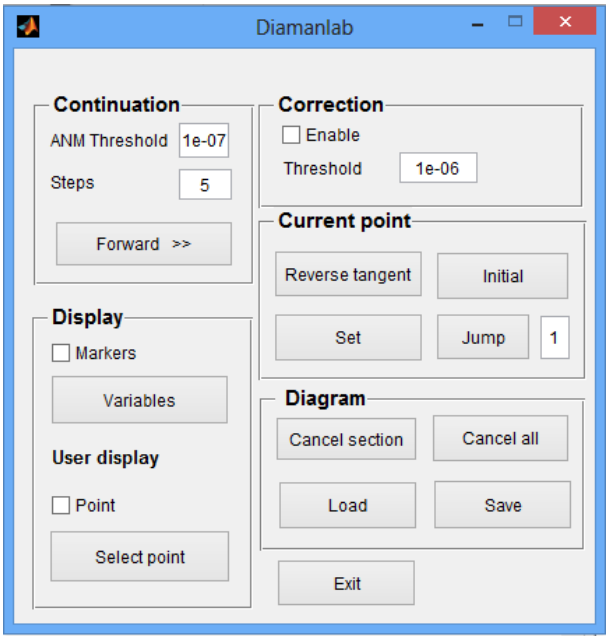

FIGURE 2. Graphical User Interface

curves, the "current point" is indicated by a square marker and an arrow figuring `U0` and `Ut`, respectively. The continuation steps appear to be performed from the representation(s) of the point `U0` in the arrow direction(s).

The interactive continuation is managed through the GUI and monitored on the projected bifurcation diagram. The GUI is divided into fives frames, namely Continuation, Correction, Current point, Display and Diagram. The frame Continuation contains the method :

- `forwardcontinuation`, *Forward* >> on the GUI, to perform the number of continuation steps indicated in the text box above.

The `ANMthreshold` is the accuracy required for the parametric representation (2). It is used to estimate the range of validity a_m and may be modified by the user.

The frame Correction contains

- `NRcorrections` to perform Newton-Raphson corrections on the "current point" when the accuracy is over the `NRthreshold` parameter. This threshold may be modified in the related text area.

The `Current Point` frame contains push buttons that act on the current point location:

- the `Reverse tangent` button reverses the continuation direction,
- the `Initial` button resets the current point to the initial guess point provided by the user,
- the `Set` button enables to capture a mouse position on the projected bifurcation diagram and to set the current point to the closest solution point,
- The `Jump` button captures a mouse position on the diagram figure. It commands a prediction-correction and update the current point position

TABLE 1. Continuation Parameters

Field name	Default value	Note
`order`	20	command level only
`ANMthreshold`	1.10^{-7}	AMM threshold
`NRcorrections`	0	NR correction off
`NRthreshold`	1.10^{-8}	NR method threshold
`NRitemax`	15	Maximum number of iterations for NR

on the bifurcation diagram. This allows the user to follow another branch of the bifurcation diagram, for instance.

Available `Display` options are:

- `Markers` to indicate (or not) checkpoints,
- `Variables` to choose the couples of components of U that are plotted as curves in the projected bifurcation diagram,
- `Point` and `Select point` to launch the user-defined display implemented in the `UserSyst` class, at each steps if the `Point` button is on, or for the point capture on the projected bifurcation diagram with `Select point`.

Display options may be set at the command level when launching a continuation, or modified through the GUI.

Four actions are operated from the `Diagram` frame. The `Load` and `Save` methods are file I/O functions to import and to export the diagram data, respectively. Diagram data are the (`params` structure, the `UserSyst` and `ContDriver` objects. The last one notably includes the checkpoint list. The `Cancel all` method allows for a reset of the checkpoint list. Particular checkpoints may be removed from the list and the bifurcation diagram, simultaneously, using `Cancel section` button and selecting the related branch sections.

.

4. Interactive continuation examples

As discussed in paragraph 3.3, Diamalab is provided with several examples. A user may either run a predefined problem, or take advantage of one of the `UserSyst` examples to implement his smooth nonlinear problem in a natural way as a R method of a new subclass of the `Syst` class.

4.1. **The `params` structure array.** A very few continuation parameters, Table 1, are involved in the ANM process. The first two are the order truncature of the series and the required accuracy for the ANM continuation. The last three are related to Newton corrections. Continuation parameters may be set at the command level when launching a continuation, or modified through the GUI.

4.2. **User-defined problem and class.** A user-defined problem may be coded implementing a class that inherits from the `Syst` class. One notices that the check for analyticity and the eventual regularization is under the user's responsibility. The user creates a class directory named `@USERSYST` in a directory, `Diamalabv1.0/EXAMPLES/USERSYST` for instance, that should be on the Matlab path. This directory contains at least two methods, the constructor `USERSYST` that indicates the number of equations of the user's system, and a function `R` containing the equations. A user-defined `disp` function (see paragraph 4.3) may be added for complementary plots. A script,

called `usersyst.m` for instance, may be implemented to run Diamanlab with user's continuation parameters and display options. Diamanlab v1.0 is provided with several examples stored in the `EXAMPLES` directory and described with detail in the user guide [11].

As an example, let us consider the Brusselator problem [21], implemented in the `EXAMPLES/Brusselator` directory, which equations are written in Fig. 1 as a `R` method. Actual residual equations may be written in a natural way using either

ALGORITHM 1: Function `R.m` for the Brusselator equations

```
function R1 = R(obj,U) % Brusselator (Seydel, 1994)
global Ck %Current order of differentiation
if isa(U,'Taylor'), R1=Taylor(Ck,zeros(6,1)); end
u1=U(1); u2=U(2); u3=U(3); u4=U(4); u5=U(5); u6=U(6); ulam=U(7);
R1(1)=2-7*u1 + u1*u1*u2 + ulam*(u3-u1);
R1(2)= 6*u1 - u1*u1*u2 + 10*ulam*(u4-u2);
R1(3)=2-7*u3 + u3*u3*u4 + ulam*(u1+u5-2*u3);
R1(4)= 6*u3 - u3*u3*u4 + 10*ulam*(u2+u6-2*u4);
R1(5)=2-7*u5 + u5*u5*u6 + ulam*(u3-u5);
R1(6)= 6*u5 - u5*u5*u6 + 10*ulam*(u4-u6);
```

vectors and matrices for better efficiency, or referring to a part of them through a subset of indexes. One notices that the function `R` may be the top function of a complex program such as usual in a finite element modeling, for instance. Inputs are a `Syst` object and a Taylor object. The output is a Taylor object containing the desired Taylor coefficients of R. The constructor method `BRUSSELATOR.m` is presented in Fig. 2.

ALGORITHM 2: Constructor method: `BRUSSELATOR.m` file

```
classdef BRUSSELATOR < Syst
% The 'BRUSSELATOR' class inherits from the Syst class
% ninc= 7 (unknows)
% This example shows the syntax to pass a parameter 'a' as argument,
% even there is no need in this example
   properties
    a=0;
   end
   methods
    function sys = BRUSSELATOR(a)
     sys = sys@Syst('ninc',7);
     sys.a=a;
    end
   end
end
```

The example is run using the `brusselator` command in the Matlab window. This opens the GUI, Fig. 2, and the diagram window, Fig. 3.a, that displays the corrected initial current point. The bifurcation diagram is built clicking on the `Forward` >> button of the GUI and controlling the results in the diagram window. Fig. 3.b shows the continuation path obtained after 20 forward steps. One notices a round marker on the path that indicates a simple bifurcation has been detected, the

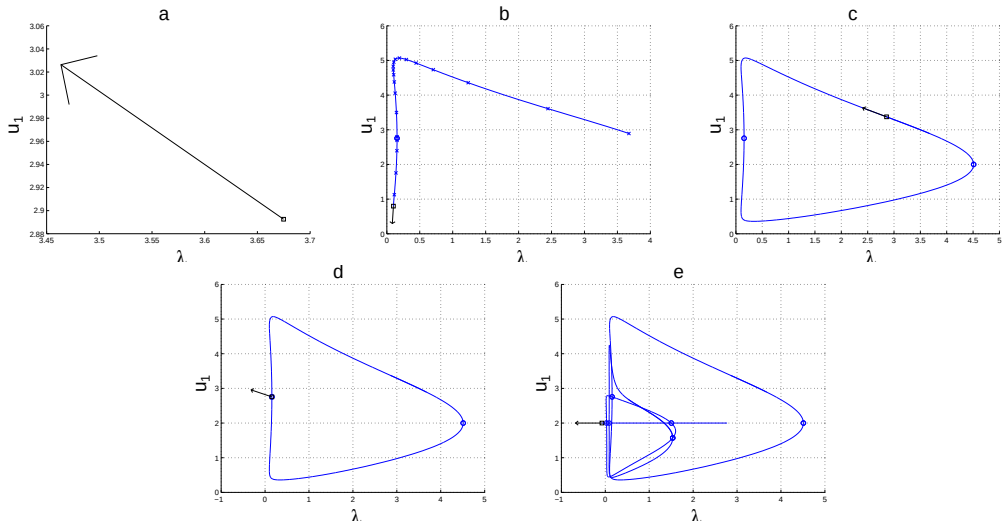

FIGURE 3. Interactive construction of the bifurcation diagram for the Brusselator system.

corresponding tangent direction is also computed. After 35 steps, Fig. 3.c, the path forms a closed loop, checkpoint markers are not plotted for a better observation of the diagram. Two simple bifurcation are detected on this continuation path.

Diamanlab allows to set the current point at a bifurcation point using the `Set` button of the `current point` frame. In an automated manner, the tangent direction `Ut` is set to the tangent direction of the bifurcated path, Fig. 3.d. The bifurcated path is followed using the `Forward` >> button. The full bifurcation diagram, Fig. 3.e, made of three interconnected loops and one straight line, is obtained repeating this procedure. It requires 120 forward steps, involving 120 Taylor series computation and 120 checkpoints.

4.3. **User display facilities.** Additional displays may be coded to analyze a solution point U. Indeed, drawing a graphical representation of U is a valuable manner to get a good insight of the computed solution. Such an interpretation feature may be used as follows:

- the user implements his own `disp.m` method in his `UserSyst` class to produce some kind of Matlab action (graphics, video, sound, ...) from the solution point U. One notices that the method has a unique argument U.
- the `disp.m` method is called either when a checkpoint object is computed (the "end" point being used as U by `disp`) or when the user selects a point in the projected bifurcation diagram using the GUI.

The Bratu example,

$$\text{Find } u \text{ such that } \ddot{u}(x) + \lambda e^{u(x)} = 0 \text{ in } (0,1),\ u(0) = u(1) = 0, \tag{5}$$

provides a basic example for such feature. The problem is discretized using a central difference scheme, what yields an algebraic system satisfying (1). Figure 4 presents the bifurcation diagram, as well as the plot of $U(x)$ for the solution point indicated with a square on the bifurcation diagram.

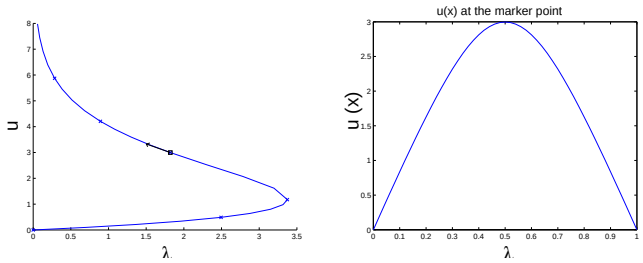

FIGURE 4. Bratu problem: (left) the projected bifurcation diagram, (right) the plot of $u(x)$ at a solution point

5. Conclusion

The interactive continuation tool Diamanlab computes solutions of a parametric nonlinear problem as Taylor-based approximations stored in checkpoints. It combines the advantages of the efficient higher-order asymptotic numerical method in its Diamant form, with AD for Taylor coefficient computations, and with object-oriented programming and GUI capabilities of MATLAB. Based on theoretical developments, our object-oriented programming involves four classes devoted to the interactive management of the continuation, to the manipulation of the generic system of nonlinear equations, to the checkpoint management and to higher-order AD, respectively. A special attention is paid to the object definition for an easy extension of Diamanlab. Future developments of Diamanlab will be oriented toward ODE system $\dot{U} = R(U)$ and boundary value problems for which stability analysis, periodic solution finding, periodic solution stability will be addressed.

Availability

Diamanlab is provided "as-is", without any express or implied warranty. The software is available to academic users at no charge under the end-user licence agreement. Licence terms and software will be downloaded from the Diamanlab's webpage `http://Diamanlab.lma.cnrs-mrs.fr`. Implementation and usage guidelines are described in the Diamanlab User Guide [11].

References

[1] E. L. Allgower and K. Georg. *Numerical Continuation Methods: an Introduction*. Springer-Verlag, New-York, 1990.

[2] R. Arquier, B. Cochelin, and C. Vergez. Manlab : logiciel de continuation interactif. Available at http://manlab.lma.cnrs-mrs.fr, Manuel utilisateur (2005).

[3] M. Berz, K. Makino, K. Shamseddine, G.H. Hoffstätter, and W. Wan. COSY INFINITY and its applications in nonlinear dynamics. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 363–365. SIAM, Philadelphia, PA, 1996.

[4] Y.F. Chang and George F. Corliss. Solving ordinary differential equations using taylor series. *ACM Trans. Math. Software*, 8:114–144, 1982.

[5] I Charpentier. Checkpointing schemes or adjoint codes: Application to the meteorological model Meso-NH. *SIAM J. Sci. Comput.*, 22:2135–2151, 2001.

[6] I. Charpentier. On higher-order differentiation in nonlinear mechanics. *Optim. Method. Softw.*, 27:221–232, 2012.

[7] I. Charpentier, A. Lejeune, and M. Potier-Ferry. The diamant approach for an efficient automatic differentiation of the asymptotic numerical method. In Christian H. Bischof, H. Martin

Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 139–149. Springer, Berlin, 2008.

[8] I. Charpentier and M. Potier-Ferry. Différentiation automatique de la méthode asymptotique numérique typée: l'approche Diamant. *C. R. Mecanique*, 336:336–340, 2008.

[9] I. Charpentier and J. Utke. Interactive continuation tools. *Optim. Method. Softw.*, 24:1–14, 2009.

[10] B. Cochelin. A path-following technique via an asymptotic-numerical method. *Computers & Structures*, 53(5):1181 – 1192, 1994.

[11] B. Cochelin and I. Charpentier. Diamanlab userguide. Available at http://Diamanlab.lma.cnrs-mrs.fr, 2013.

[12] B. Cochelin, N. Damil, and M. Potier-Ferry. Asymptotic-numerical methods and Padé approximants for non-linear elastic structures. *Int. J. Numer. Meth. Eng.*, 37:1187–1213, 1994.

[13] B. Cochelin, N. Damil, and M. Potier-Ferry. *Mthode Asymptotique Numrique*. Hermes Science Publications, Paris, 2007.

[14] E. Doedel, H. Keller, and J.P. Kernevez. Numerical analysis and control of bifurcation problems (i) bifurcation in finite dimensions. *Int. J. Bifurcat. Chaos*, 1:493–520, 1991.

[15] W. Govaerts and Y. Kuznetzov. Interactive continuation tools. In B. Krauskopf, H.M. Osinga, and J. Galan-Vioque, editors, *Numerical Continuation Methods for Dynamical Systems*, chapter 2, pages 51–75. Springer Netherlands, 2007.

[16] A. Griewank, D. Juedes, and J. Utke. ADOL–C, a package for the automatic differentiation of algorithms written in C/C++. *ACM T. Math. Software*, 22(2):131–167, 1996.

[17] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.

[18] Andreas Griewank and Andrea Walther. Treeverse: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Software*, 26:200–0, 1997.

[19] Y. Koutsawa, I. Charpentier, E.M Daya, and M. Cherkaoui. A generic approach for the solution of nonlinear residual equations. Part I: the Diamant toolbox. *Comput. Method. Appl. M.*, 198:572–577, 2008.

[20] J. Pryce and J. Reid. AD01, a Fortran 90 code for automatic differentiation. Technical Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 1998.

[21] R. Seydel. *Practical Bifurcation and Stability Analysis*. Number 5 in Interdisciplinary Applied Mathematics. Springer, 3rd edition, 2009.

[22] J.M.T. Thompson and A. Walker. The nonlinear perturbation analysis of discrete structural systems. *Int. J. Solids Struct.*, 4:757–768, 1968.